CS 231n

Lecture 1

50% of neurons for visual cortex.

History: → Face Detection by (Viola) → Adaboost Algorithm for Real time F.D.

→ SIFT Feature Idea is to match and the entire object eg. a stop sign to another stop sign is very difficult because of angles, occlusion, viewpoint etc. But some features, some parts that tend to remain invariant to changes. So the task obj. rec. began with identifying these critical features & match the features to a similar object.

→ Human Reco. - Histogram of Gradients (HoG)
Deformable Part Model

2000 - Benchmark data set (First) PASCAL
20 object class

→ ImageNet → 22k categories & 14M images

→ ILSVRC → 1000 classes & 1.4M images

Learn
Image Classification
Object Detection
Image Captioning

Lecture 2. May 22, 2018

Image Classification.

Image is big number.

Challenges: Viewpoint, Illumination, Deformation | Occlusion, Background Clutter, Intraclass variation

Rather than writing hard code approach, take Data driven approach.

1) Collect a dataset of images & labels.
2) Use ML to train a classifier
3) Evaluate the classifier on new images.

Nearest Neighbor

```
def train(images, labels):
    # ML
    return model
```
} → Memorize all data and labels

```
def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
} Predict the label of the most similar training images

CIFAR-10

10 classes.
50K training images
10k test images.

How to Compare?

L1 Distance

test image - train image → pixel-wise abs. value difference

```
def train
    self.Xtr = X   } simply remember train O(1)
    self.ytr = y   } all data      predict O(N)
def predict(self, X):
    :
    )
```

K-Nearing Neighbor Algo.

Majority from k

L1 & L2 Distances

Setting Hyperpara.

Train & test — Bad ✗

| train | test |

Idea1 Peaking while training

Better ✓

Idea2 | train | val. | test |

Idea3 Cross Validation

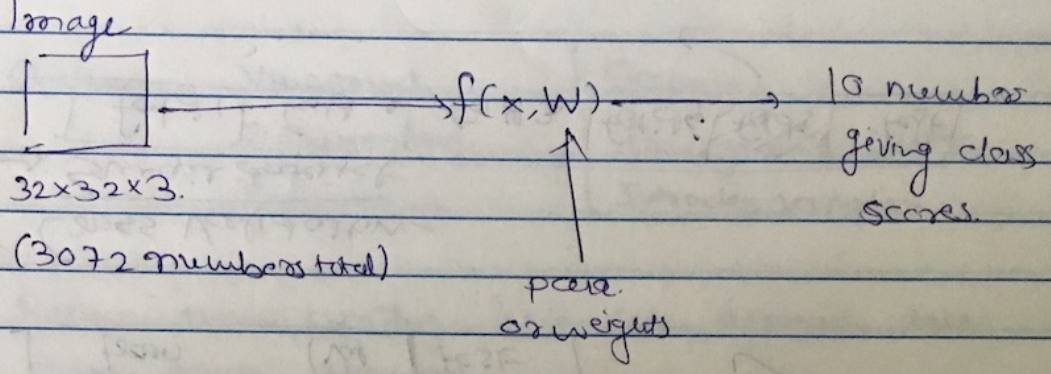
→ Small dataset

| fold1 | fold2 | fold3 | fold4 | fold5 | test |

KNN. Never used.

Linear Classification.

Building blocks of NN.



$$f(x, W) = Wx + b$$

10×1 | 3072×1 10×1

10×3072.

Weights are like templates for all classes
Just like a template matching
~~One template per category~~

Que How can we tell whether this W is good or bad?

Next lecture

Loss

Optimization

Lecture 3 is in

05/04/19

CS-231n CNN 05/24/18

Lecture-3

Till Now.

Image Classification.

Problems

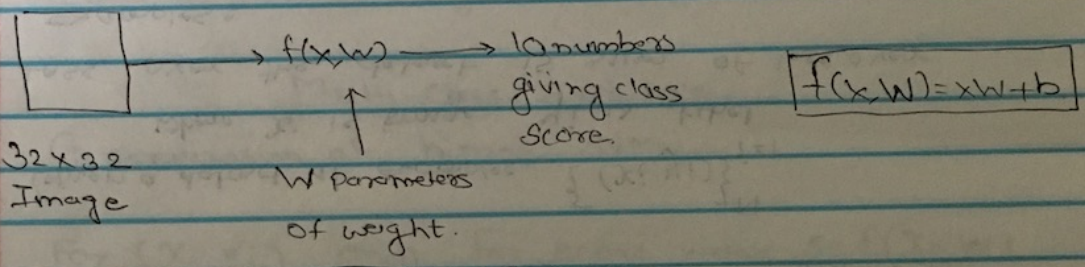
CIFAR-10.

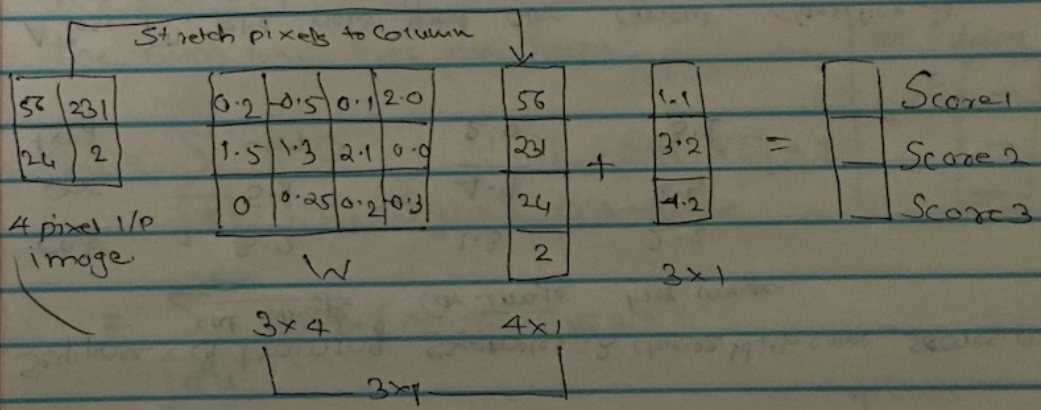
K-Nearest Neighbor

Train / Test / Validation / Cross-Validation

Hyperparameters.

Linear Classification.





W → no. of classes x (pixels ~~flatten~~ stretch into column)

eg. 10 x 3072 (For CIFAR-10)

Once train, if we unravel each row in 32x32x3 then we can visualize weight template for that class.

Que: Which W should be best?

We use training data for this.

How bad W is quantitatively by looking at the scores is a loss function

After getting the badness of Ws, we need to come up with some value of Ws that are least bad and this is called Optimization

Suppose 3 training examples, 3 classes. With some, scores are:

	Cat Image	Car Image	Frog image
cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

A loss fn. tells how good our current classifier is.

Given a dataset of examples $\{(x_i, y_i)\}_{i=1}^{N}$

where x_i is image y_i is label.

Loss over the dataset is sum of loss over examples:

$$L = \frac{1}{N} \sum_i \underbrace{L_i(f(x_i, W), y_i)}_{\text{Loss for 1 example}}$$

Pred. fn.

Pred. fn.

takes the value x and weight W and makes some prediction of y. (10 numbers).
Then we will define loss fn Li, which will take the predicted scores that coming out of fn f together with true target label y and give us some quantitative value how bad those predictions are for that training example. Now, the final loss L will be the average of these losses

Details about Loss function

Multi-Class SVM Loss

Generalization of binary SVM.

For (x_i, y_i) and for score vector $s = f(x_i, W)$ the svm loss has the form:

for 3 classes

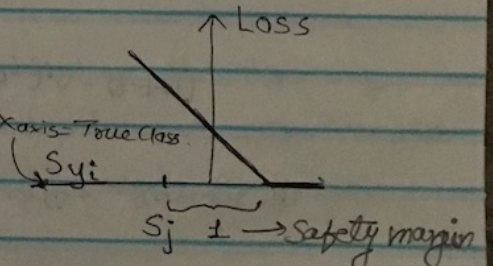
$$s = \begin{bmatrix} - \\ - \\ - \end{bmatrix}$$

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} > s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise.} \end{cases}$$

s_{y_i} → score for the true class for i^{th} example

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Also known as "Hinge Loss"



As score for true class ↑, loss ↓. If score is > margin loss = 0.

In a 3 training example.

L_i = max(0, 5.1 − 3.2 + 1) + max(0, −1.7 − 3.2 + 1)

Cat = max(0, 2.9) + max(0, −3.9)

= 2.9 + 0

= 2.9 ← how much our classifier screwed up.

L_i = max(0, 1.3 − 4.9 + 1) + max(0, 2.0 − 4.9 + 1)

Car = 0.

L_i = 12.9

Frog

Loss over full dataset

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

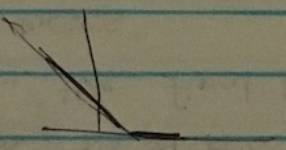
= (2.9 + 0 + 12.9) / 3 = 5.27

1 is an Arbitrary Choice.

Some intuitions:

→ Min loss we can get is 0.

→ Max loss " " " ∞



→ All Ws are initialized to 0 in beginning then loss will be no. of classes − 1

→ We can use Squar hinge loss also.

Vectorized implementation

```
def L_i_vectorized (x, y, W):
    scores = W.dot(x)
    margins = np.max(0, scores - scores[y] + 1)
    margins[y] = 0.
    loss_i = np.sum(margins)
    return loss_i
```

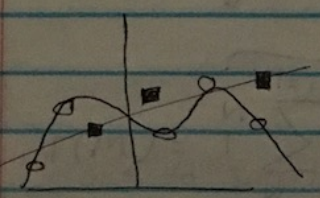
Que
If we found a W such that L=0
Is this W unique?

$$f(x, W) = W \cdot x$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max\left(0, f(x_i, W)_j - f(x_i, W)_{y_i} + 1\right)$$

→ No

There are other choices of W (like 2W) which satisfy the condition of L=0.
How to choose which W to use?
This situation is because, we have defined loss fn in terms of the data & we have only told our classifier that it should try to find the W that fits the training data.
In reality, we don't care about fitting the training data. We apply classifier which is trained on test data.
An example out of L.C. and general to ML concept



If we only ask our classifier to fit training data, then we might get the curve shown in fig. (perfectly classify all the train data) BAD!!!
Won't work well on new data.

"How to solve this problem?
Regularization

Regularization

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple" so it works on test data

Reg. encourages to pick "simple" Ws.
Now we have ① Data loss
② Regularization loss.

λ is hyper-parameter, which we need to tune.

So, with these Reg. terms included we encourage the model to take (consider) smaller polynomial terms (simpler) such that the wiggle curve → straight line

If the model is complex then this is the penalty we use to make it simple. We need to use this penalty for using model's complexity.

Many Reg. available. Most common are

L2 Reg $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 Reg $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2)

Max norm Reg,

Dropout

(Batch normalization, stochastic depth)

Que. How does the L2 Reg. measure the complexity of the model?

Eg. Let say we have two Ws for same data x

$W_1 = [1 \ 0 \ 0 \ 0]$

$W_2 = [0.25 \ 0.25 \ 0.25 \ 0.25]$

$x = [1 \ 1 \ 1 \ 1]$

For L.C. we will get same dot product.

Which W the L2 Reg. will choose then?

It will choose W2. because it has smaller norm.

L2 Reg. prefers to spread the influence across all the diff. value of x.

For L1 reg, we would prefer W_1 over W_2. It has reverse interpretion. It has diff. notion of complexity

Multinomial logistic Regression (Softmax Classifier)

Cat Image

cat 3.2

Car 5.1

frog -1.7

We have meanings of scores unlike SVM loss.

Scores = unnormalized log probabilities of the classes

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i, W)$$

Softmax function

Want to maximize the log likelihood, or (for a loss f°.) to minimize the negative log likelihood of the correct class.

$$L_i = -\log P(Y = y_i | X = x_i)$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

eg	cat	3.2		24.5	0.13
	car	5.1	exp →	164.0	normalize → 0.87
	frog	-1.7		0.18	0.00
				unnormalized probabilities. (All +ves)	probabilities.

$$L_i = -\log(0.13)$$
$$= 0.89.$$

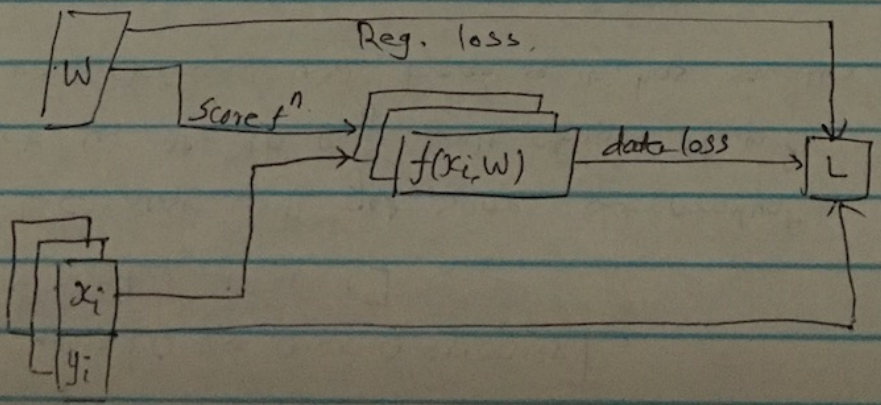
Some Intuition

Min loss → 0. → To get this score of correct class → ∞

Max loss → ∞ → $-\log(0) \rightarrow -(-\infty) \rightarrow \infty$

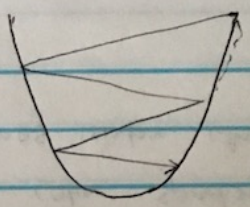
We never get this min/max values with finite precision.

If all the $s \approx 0$ then loss → $-\log\left(\frac{1}{C}\right) \rightarrow \log(C)$



Que: How do we actually find this W that ↓ the loss?

Optimization



Strategy 1 Random Search. BAD.
Strategy 2 Follow the slope. ✓

In 1-D

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In - Multi-D, the gradient is the vector of partial derivatives along each dimension.

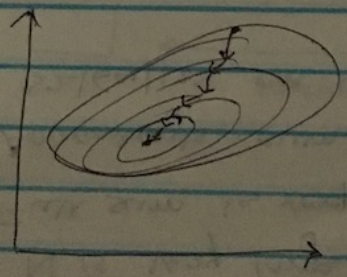
The slope in any direction is the dot product of the direction with the gradient. The direction of steepest descent is the -ve gradient.

Numeric Gradient → approx., slow.
Analytic Gradient → exact, fast.

Gradient Descent:

```
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += -step_size * weights_grad  # Parameter Update.
```

hyperparameter



Other fancier update rules
1. G.D with momentum
2. Adam Optimizer.

SGD

N is very large value. For Imagenet dataset N is 1.3m. Full sum is very expensive in this case.

Approximate sum using a minibatch of example 32/64/128 are very common.

```
While True:
    data_batch = Sample_training_data(data, 256)
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += -step_size * weights_grad
```

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

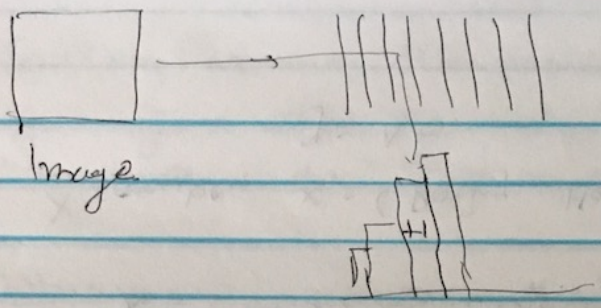
Gradient is linear f^n and sum of the gradient of the losses for each individual terms. So we have to iterate over the entire dataset.

Hack is to use Stochastic G.D.
We use mini-batch to estimate gradient.

Image Features

Not a good idea to feed raw pixels to linear classifier.
Find features. Concatenate. Feed to L.C.
Before N.N. there were few of them.

① Color Histogram: Divide into hue color spectrum. Count for each pixel that comes under which spectrum.



Another common feature vector that we saw before NN.

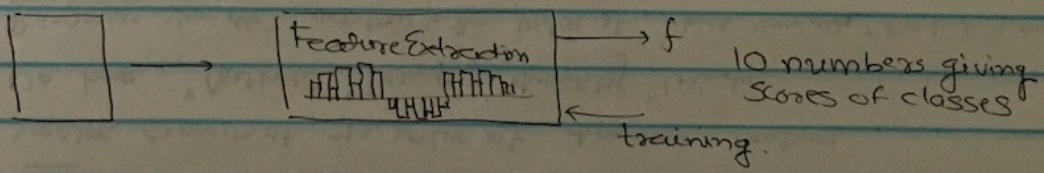
Histogram of oriented gradients (HoG)

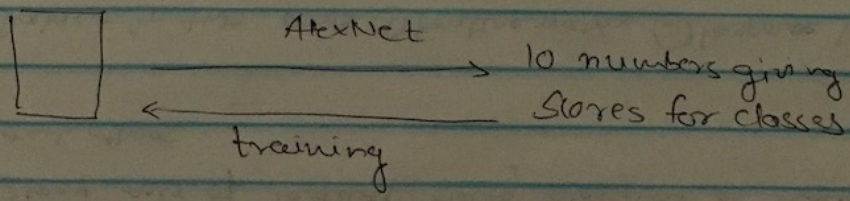
Divide image into 8×8 pixel regions. Within each region quantize edge direction into 9 bins.

If we have 320×240 then we will have 40×30 bins. Each bin is quantized with 9 numbers ⇒ Feature vectors are $40 \times 30 \times 9 = 10800$.

Color histogram gives what color presents in the image & HoG gives what type of edges exist in the image.

Image Features vs ConvNets





Lecture 3 is in other Notes. Bring it here!!! 05/24/18.

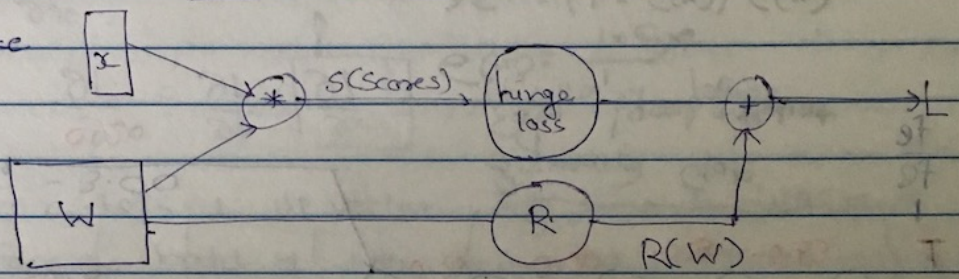
Lecture 4 05/24/18.

Backpropogation and Neural Network

Analytic Gradient Calculation for Complex Function using Computational Graph.

$f = Wx$

like



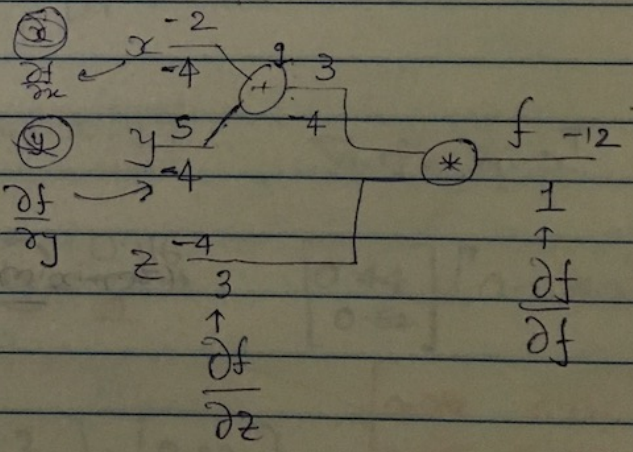
We can use B.P. with Computational Graph

$f(x,y,z) = (x+y)z$

$x = -2$

$y = 5$

$z = -4$



$q = x+y$

$\frac{\partial q}{\partial x} = 1$ $\frac{\partial q}{\partial y} = 1$

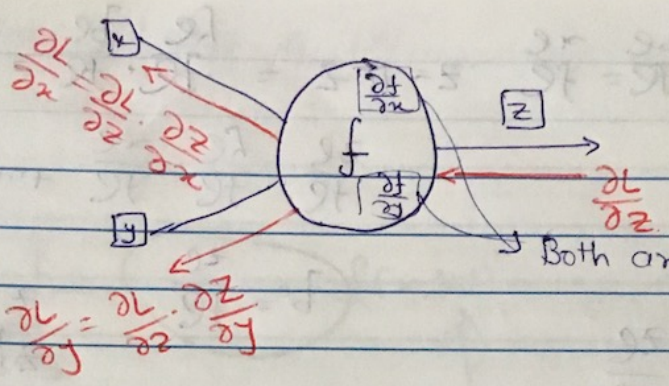
$f = qz$

$\frac{\partial f}{\partial q} = z$ $\frac{\partial f}{\partial z} = q$

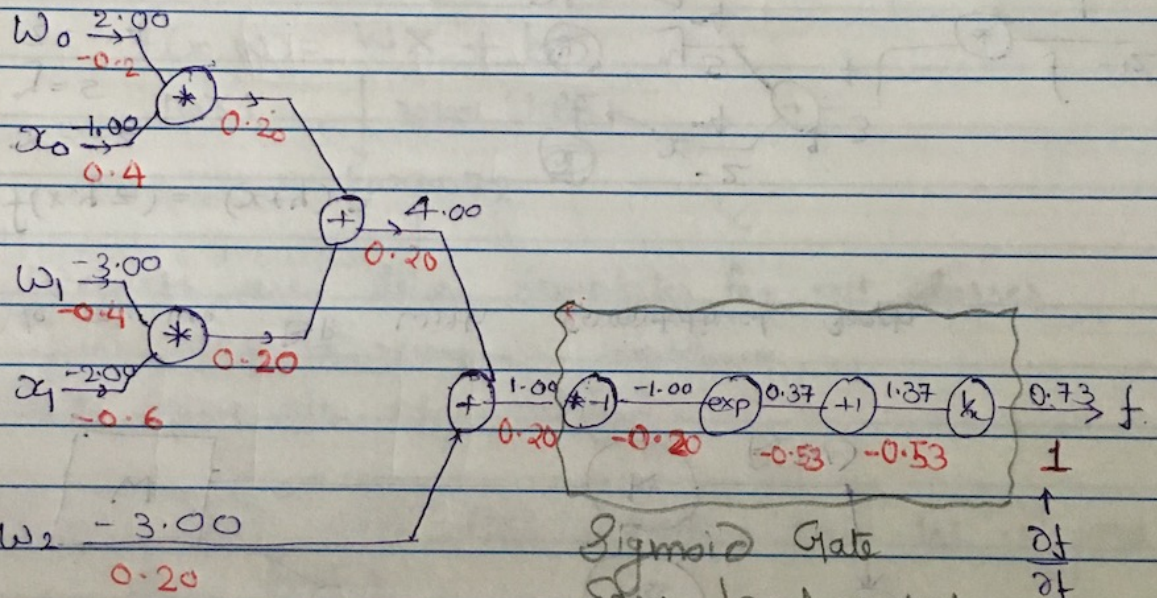
We want $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial f}{\partial y} = z \cdot 1 = z$ $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} = z \cdot 1 = z$

Eg. 1



Eg. 2 $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

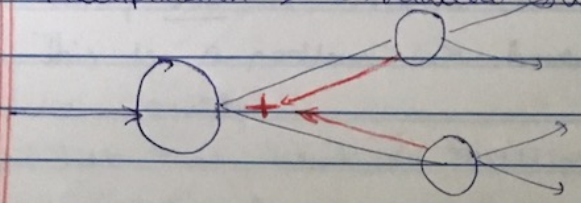


Add → Gradient Distributor

Add → Gradient Distributor

Max → Gradient Router.

Multiplication → Gradient Switcher.



Gradient Adds at branches

For Vectors

$\frac{\partial f}{\partial x}$ is Jacobian Matrix.

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial f} \cdot \left[\frac{\partial f}{\partial x}\right]$$ It's very large.

For 4096-d ile vector J.M size is 4096×4096.

For a batch of 100, it will be 409600×409600 !! Huge

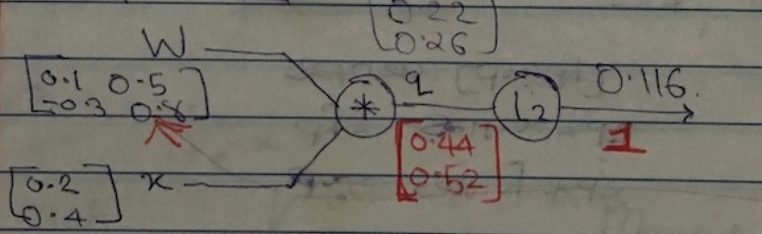
But it's just a diagonal !

A vectorized example:

$$f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

L2 of W.x

Comp. Graph



$$\nabla_W f = 2q \cdot x^T$$

$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \end{bmatrix}$$

$$\begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix} \quad q = W \cdot x = \begin{bmatrix} 0.02 + 0.2 \\ -0.6 + 0.32 \end{bmatrix} = \begin{bmatrix} 0.22 \\ 0.26 \end{bmatrix}$$

2×2 2×1

$$\begin{bmatrix} 0.088 & 0.154 \\ 0.176 & 0.308 \end{bmatrix}$$

$$\nabla_x f = 2W^T \cdot q$$

$$f(q) = q_1^2 + \ldots + q_n^2 = (0.22)^2 + (0.26)^2 = 0.116$$

$$\frac{\partial f}{\partial q_i} = 2q_i \qquad \frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \sum (2q_k)(1_{k=i} x_j) = 2q_i x_j$$

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x
        self.y = y
        return z  #Must keep these values!!

    def backward(dz):
        dx = self.y * dz
        dy = self.x * dz
        return [dx, dy]
```

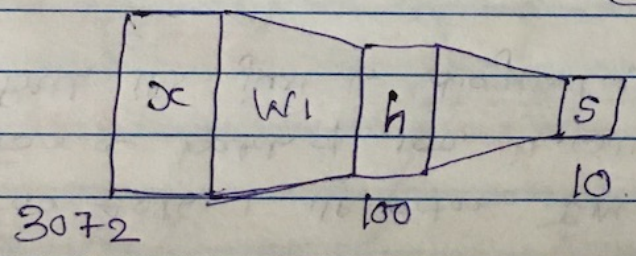
Neural Network

So far

Linear score f^n $f = Wx$

Now, 2-layer N.N $f = W_2 \max(0, W_1 x)$

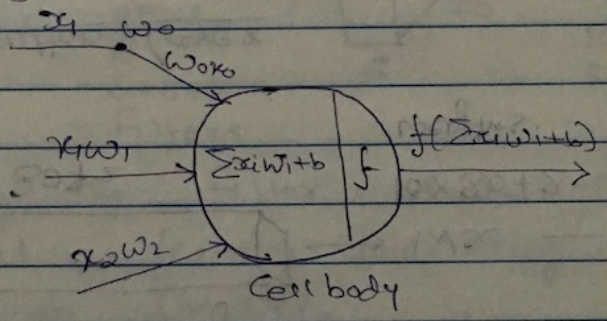
Very imp non-linearity



Simpler functions stacked on each other in a hierarchial way in order to make up a more complex non-linear function.

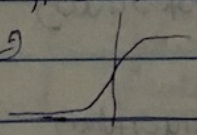
In the template analogy, we saw that we had only one template per class in linear classifier (eg. fuzzy "red" car template for Car).
This is a problem. We have all colors / modes of car in reality.
What this kind of multiple nlw lets you do is, each of the intermediate variable h_i, W_1 can still be these kind of templates but now we will have all of scores for these temp. in h_i and we can have another layer on top.

Basically, W_1 is now many templates and W_2 is weighted sum of these templates.



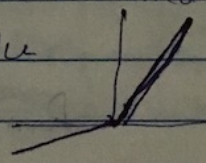
Many types of f: (Activation functions) (Non-linearity)

Sigmoid

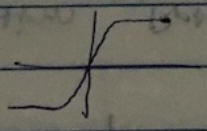


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Leaky Relu



tanh

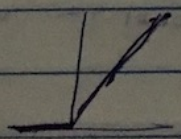


$\tanh(h)$

Maxout

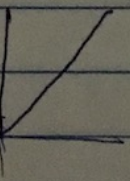
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU



$\max(0, x)$

ELU



$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

Lecture 5

05/24/2018.

Convolution Neural Network —

Yan Lecunn → 1998 LeNet

Alex Krizhevsky → 2012 Alexnet.

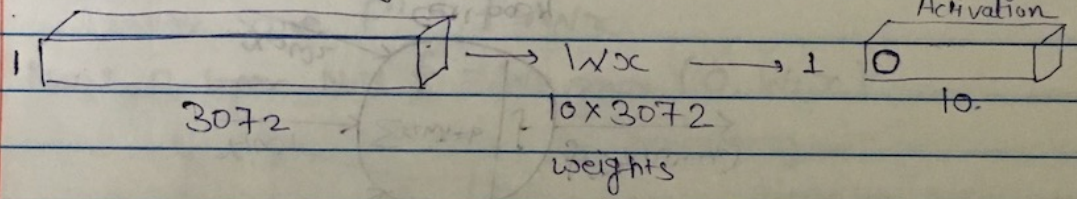
Classification, Retrieval, Detection, Segmentation

Pose Recogni.

Image Captioning → Sentense descr. of image.

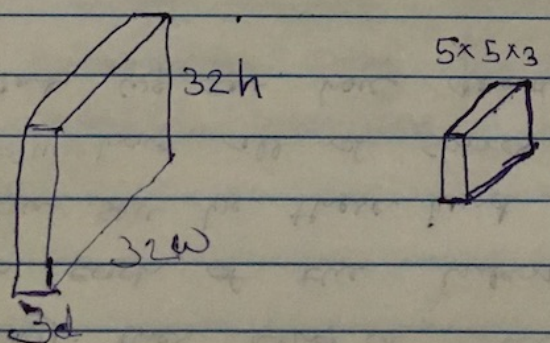
Fully Connected Layer

32 × 32 × 3 image → Stretch to 3072 × 1



Convolution Layer

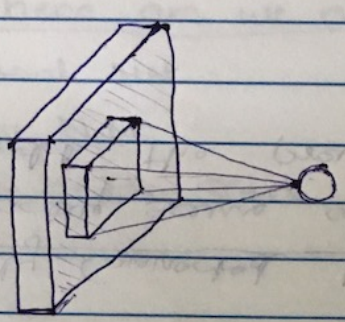
It preserves Spatial structure. We do not stretch the image 3D input in our case 32 x 32 x 3



Weights are going to be small filters.

Convolve the filter with the image. Slide over the image spatially and computes dot product.

Filter always extend the full depth of the input volume



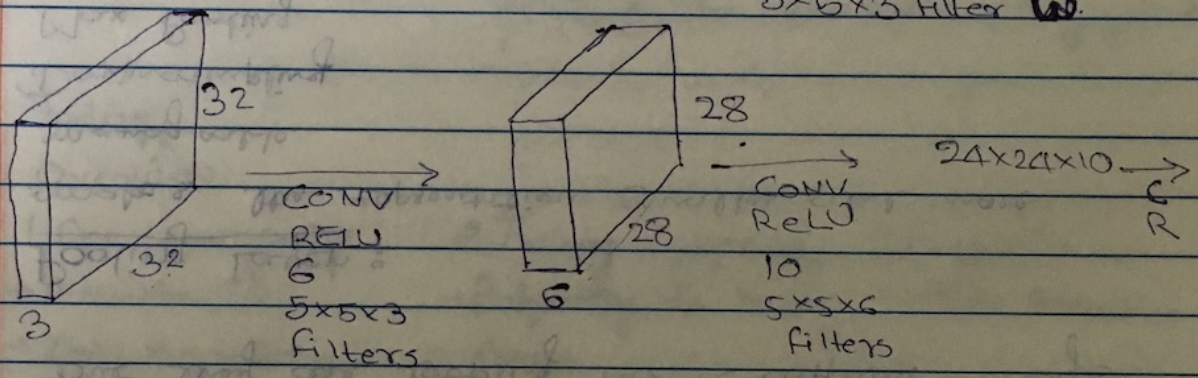
1 number

Total 75 computation (Dot product) + bias

$w^T \cdot x + b$

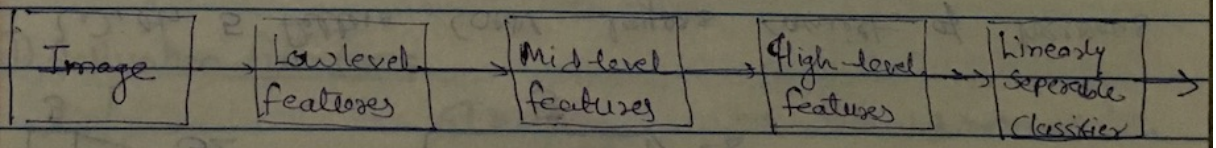
32×32×3 image

5×5×3 filter w



Each of the filter produces activation map.

We end up learning hierarching of filters, when stacked together in ConvNet.



Edges etc. Corners/blobs etc

Slide

Strides

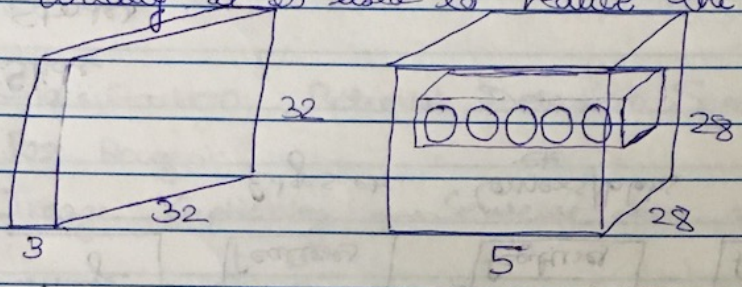
Padding

9

1 × 1 Convolution

Each filter has size 1 × 1 × a and it performs a-dimensional dot product.

Usually it is used to reduce the spatial dimension



With 5 filters, CONV layers consists of neurons arranged in a 3D grid (32 × 32 × 5)

There will be 5 different neurons all looking at the same region in the input volume but they are looking for a different things

Pooling Layer:

makes the representation smaller and more manageable

Downsampling

Max Pooling

No change in depth.

Fully Connected Layer

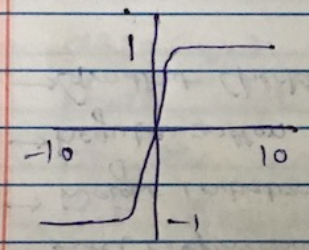
Exactly same as explained before

Stretch the previous layer

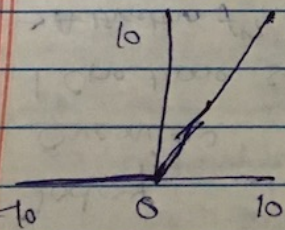
This gives very inefficient gradient update.
Two dimensional W example is given in above graph.
Our hypothetical optimal direction is 45° W-S as shown but we cannot go in that direction directly. We have to take the zigzag path as those two are the only dirn. in which we can take steps (descent).

So we need zero-centered values of x_i.

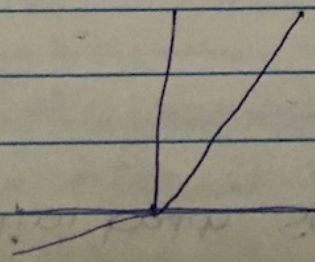
→ exp() is a bit compute expensive.



tanh
Squashes [-1, 1]
Zero centered
Still kills gradients when saturated. :-(



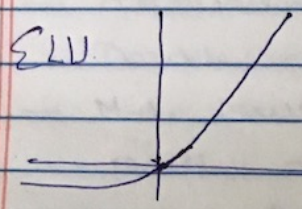
ReLU $f(x) = \max(0, x)$
does not saturate (in +ve region)
Comp. efficient
Converges much faster than sigmoid.
Killing the gradient in half regime.



Leaky ReLU $f(x) = \max(0.01x, x)$
does not saturate
Will not die.

Parametric ReLu $f(x) = \max(\alpha x, x)$

↑ Not Constant



All benefits of ReLu

Closer to zero mean o/ps

-ve saturation regime compared with Leaky ReLU.

adds some robustness to noise

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(\exp(x) - 1) & x < 0 \end{cases}$$

TLDR

→ Use ReLU

→ Try Leaky ReLU/ELU

→ Try tanh

→ Don't use Sigmoid

(2) Data Preprocessing

General idea → zero-centered. (We know the reason)

→ Normalization. (All features are in same range & contribute equal)

In Practice for images:

1) Subtract the mean image (AlexNet)
(mean image = [32, 32, 3] array)

or 2) Subtract per-channel mean (VGGNet)
(mean along each channel = 3 numbers)

(3) Weight Initialization

What if all $W = 0$ initially?

All the neurons will do the same thing. All will do same thing. & so all are going to get same gradient.

No symmetry braking here. All will learn same thing!!!

First Idea: small random number.
(Gaussian with zero mean and std = 0.01).

In the backward pass everything is getting collapsed again and again as we multiply upstreams gradients with local (W is small). And Gradients collapsing to zero as well

What if, if we choose big weights? 1 instead of 0.01? All neurons completely saturated, either -1 or 1. Gradients will be all 0. (Here we have tanh activation)

Rule of thumb for initialization

→ Xavier Initialization

→ We are going to scale by the number of inputs that we have

④ Batch Normalization

We want unit Gaussian activations? Just make them so

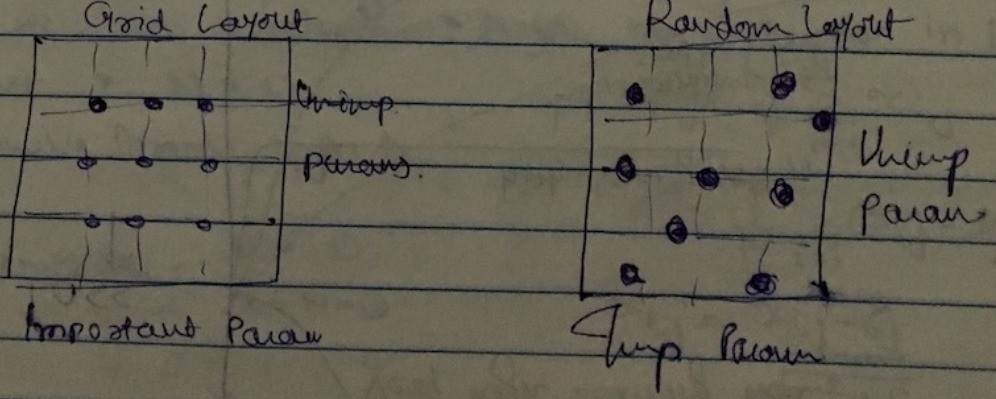
⑤ Babysitting the learning process

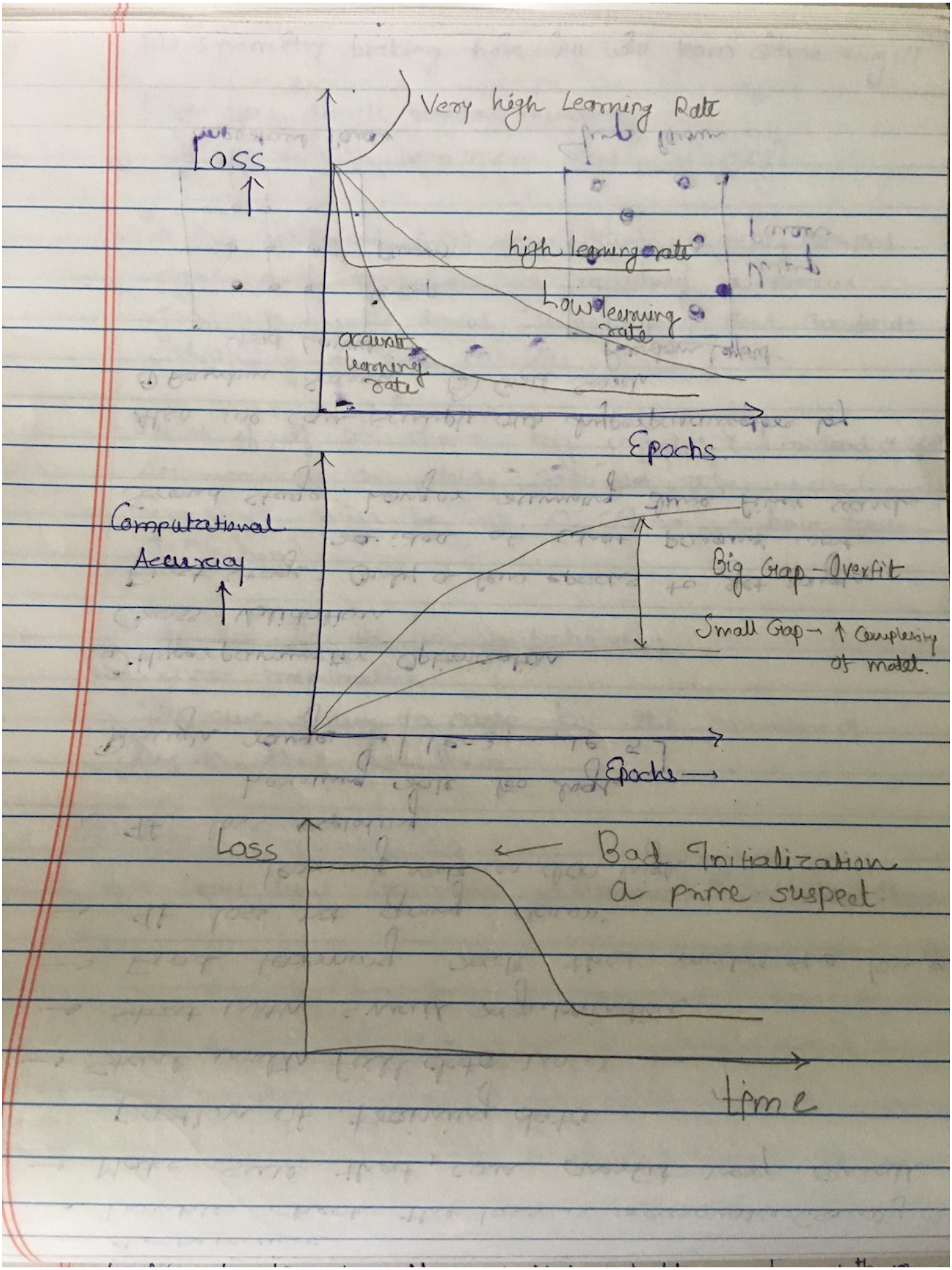
→ Data Preprocess (mean, norm).
→ Architecture
→ Double check the loss is reasonable. (Sanity check)
→ Make sure that, can overfit very small portion of training data.
→ Start with full data now.
→ Start with small regularization
→ Find learning rate that makes the loss go down.
→ If loss not going down:
learning rate is too high
If loss exploding:
learning rate too high.
Rough range → [1e-3, ..., 1e-5]

⑥ Hyperparameter Optimization

Cross-Validation
First Stage: Only a few epochs to get rough idea of what params work.
Second Stage: longer running time, finer search.

Also, we can sample our hyperparameters by
① Random Search ② Grid Search.




Very high Learning Rate
Loss
high learning rate
Low learning rate
Accurate learning rate
Epochs
Computational Accuracy
Big Gap - Overfit
Small Gap → ↑ Complexity of model.
Epochs →
Loss
← Bad Initialization a prime suspect
time

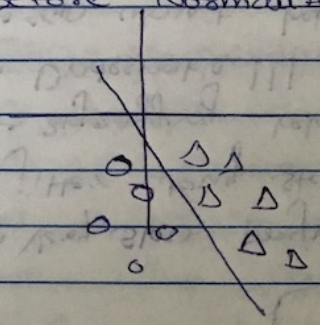
Lecture 7: Training Neural Network II June 10th 18.

One good point

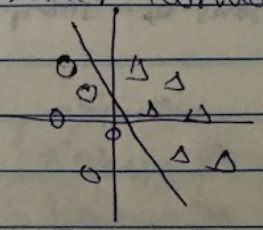
It is very common in CNN to zero mean & normalise the data so it has zero mean & unit variance.

Some extra intuitions here:

Before Normalization



After Normalization



In the simple case of binary classification, we can draw a seperation line even if the data are not normalized. (left) but now if that line wiggles just a little bit. then the classification will get totally destroyed.

Loss fn is very sensitive to weight matrix & learning will become difficult.

But in right, if it is zero-centered with unit variance then loss is less sensitive to weight matrix.

So we have implemented Batch Normalization in L6.

Input $x: N \times D$

Learnable Params: $\gamma, \beta: D$

Intermediates: $\mu, \sigma: D$, $\hat{x}: N \times D$

Output $y: N \times D$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j = \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

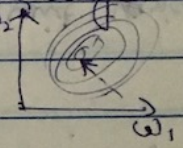
$$y_{i,j} = \gamma_j \hat{x}_{ij} + \beta_j$$

Topic ① Optimization
② Regularization
③ Transfer Learning

Loss shows how bad or good our weight are.

Simplest Optimization we have seen is SGD

```
While True:
    weights_grad = evaluate_gradient(loss_fn, data, weights)
    weights += -step_size * weights_grad
```

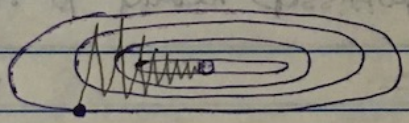


This simple soln has some issues in practice:

[1] What will SGD do when loss changes quickly in one dirn & slowly in other?

i.e. loss is very sensitive to w_2 & less to w_1

like



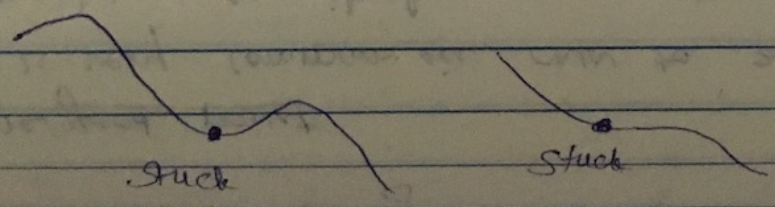
→ Very slow progress along shallow dimension (↔) and jitter along steep direction (↕)

→ Zig zagging behaviour.

→ Undesirable !!!

→ SGD won't behave nicely if largest & smallest is very large that too in million-D dimension

[2] What if loss fn has local minima or saddle point?



3. If we are getting noisy to the gradient at every point, and if we run SGD with this messed up gradients so if there is a noise in the gradient estimates, then vanilla SGD kind of meander around the space and might take longer to get minima.

Adding a momentum term can address the above problems

SGD + Momentum

- We measure velocity over time.
- We add gradients to the velocity
- And we step in direction of velocity rather than stepping in dir^n of grad.
- Here, ρ corresponds to friction. Used to decay the velocity by the friction constant rho.

Equations.

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

$v_x = 0$.

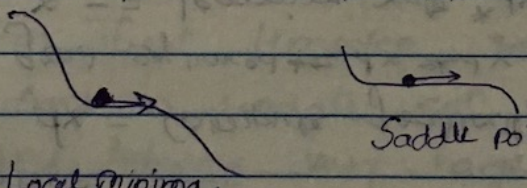
While True:

dx = compute_grad(x)

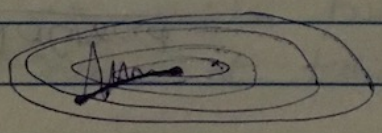
vx = rho * vx + dx

x += learning_rate * vx

This helps in all problems above.



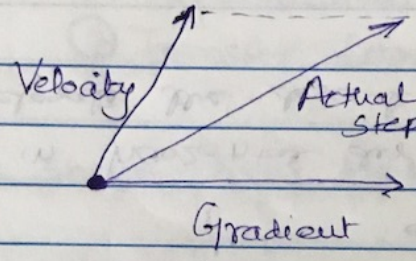
Ball picking up the speed & roll down
The point will have velocity ~~when~~ we pass local min. even if it doesn't have grad



↑ In this, zigzags will cancel out each other by using momentum

And in horizontal dir^n, our velocity will keep ↑ & will accelerate the descent.

One picture for SGD + Momentum



→ We take weighted avg. of these two.
→ Helps overcome noise.

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Q1: What happens with AdaGrad? What does this scaling do?

For two coordinates, one with high grad & one with small then we add the sum of the squares of the small gradient, we are going to divide by a small number and accelerate the movement along the slow dim^{n}. For large grad we are dividing by a large number & slower down the process along wiggling dim^{n}.

Q2: What happens to step size over long time?

Step size gets smaller & smaller as we are adding to squared term & dividing by

We might get stuck at a saddle point with AdaGrad.

RMSProp

Addresses above concern

grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dw * dw

We decay the grad_squared instead of its accumulating

Adam

Maintain estimate of first moment and second moment

Takes advantage of both momentum (velocity concept) and Adagrad (RMSProp (Gradient Squared Concept))

→ Problem in first timestep!

For first timestep second moment → 0.

And in update eqn we are dividing by ~0 value and taking very large step. So, initialization is completely messed up, & cannot converge.

→ Adam adds bias term to avoid this problem of taking very large steps.

All opt. algo. has learning rate as a hyperparameter

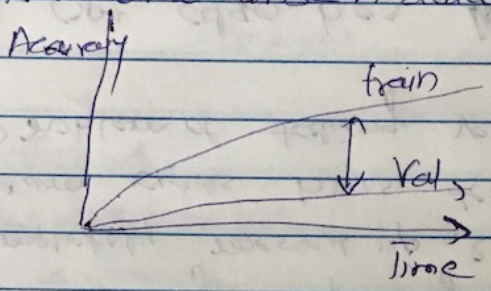
→ Learning rate decay over time!

Step decay: decay by 1/2 every few epochs

exponential decay: $\alpha = \alpha_0 e^{-kt}$

1/t decay: $\alpha = \frac{\alpha_0}{1 + kt}$

All these strategies reduce training error.
All opt. helps us driving down " ".
But we do not care about training error much.
We really care about reducing the gap between train and unseen data.



Soln

① Model Ensambles
→ Train multiple models
→ Avg. their results at test time

How to improve Single-model performance?

Regularization

Common Use

L2	$R(W) = \sum_k \sum_l W_{k,l}^2$
L1	$\sum_k \sum_l \lvert W_{k,l} \rvert$
Elastic Net (L1+L2)	$\sum_k \sum_l \beta W_{k,l}^2 + \lvert W_{k,l} \rvert$

Dropout

Randomly set some neurons to zero.
Prevent Overfit.
Network does not depend on any of the nodes
Distribute the idea across many different features
Kind of doing ensemble in single model

At test time prediction f^n, just add one more multiplication by dropout probability

Better generalization.

Data Augmentation

Increase data

Flips, Crops, brightness, contrast

Transfer Learning

We need lot of data to avoid overfitting.
But usually we do not have large data. Also it takes long for training with large dataset.

	Similar Dataset	Very different dataset
Very little data	Use linear Classifier on top layer	Try linear classifier from different stages!!
Quite a lot of data	Finetune a few layers	Finetune a large number of layers

Lecture 8 Deep Learning Software June 11th 2018

CPU / GPU

Deep Learning Framework
- Caffe / Caffe2
- Theano / TensorFlow
- Torch / PyTorch

GPU - Nvidia - CUDA Abstraction
Higher level API - CUBLAS, CUFFT, CUDNN

Model is on GPU. (Weights). Data are sitting on the hard drive.

Deep Learning Framework

Caffe → Caffe2

Torch → PyTorch

Theano → Tensorflow

Others
Paddle
CNTK
MXNet

Numpy can't run on GPU. Also we have to calculate Gradients manually in Numpy.

Lecture 9 CNN Architectures

June 15th 2018

Case Studies
- AlexNet
- VGGG
- GoogleNet
- ResNet

Also,
- Network in Network
- Wide ResNet
- ResNeXT
- Stochastic Depth
- DenseNet
- FractalNet
- SqueezeNets

Case Study

AlexNet:

Conv1 → MaxPool1 → Norm1 → Conv2 → MaxPool2 → Norm2 → C3 → C4 → C5 → MP5 → FC6 → FC7 → FC8

Input → 227 × 227 × 3

First (C1) → 96 11 × 11 filters applied at Stride 4.

Output volume size of this → (227 − 11)/4 + 1 = 55

⇒ 55 × 55 × 96.

No. of parameters: 11 × 11 × 3 × 96.

Second (P1) : 3 × 3 filters applied at Stride 2.

Output volume size of this → (55 − 3)/2 + 1 = 27

⇒ 27 × 27 × 96

No. of parameters: 3 × 3 × 3 × 96. (Tricked !!!)

Pooling layer has no parameters (Of course!!)

ZFNet:

Just improvement of AlexNet in hyperparameters.

VGGNet → Small Filters, Deeper Network.

16 → 19 layers

Only 3 × 3 Conv layers

Why smaller filters?

Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer.

But deeper, more non-linearities.

And fewer parameters: $3*(3^2C^2)$ vs. 7^2C^2 for C channels per layer.

One thing to notice is that most of the parameters are in initial Conv layers (due to large spatial size) and last FC layers.

→ Total memory for 16 layer VGGNet ~24M x 4 Bytes ~ 96M
96M / image / forward pass
~ 100MB for 1 image for single pass !!! Huge !!
*2 with Backward pass

→ Total Params: 138M

GoogLeNet

Deeper Network

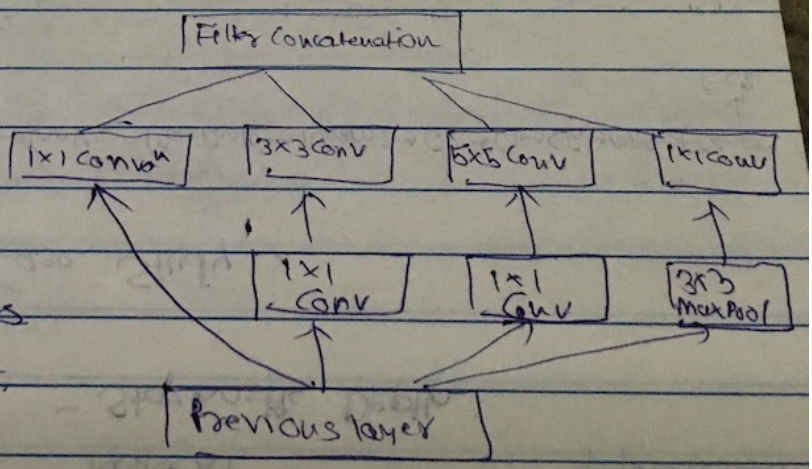
22 layers

Inception Module

No FC layers

Only 5M parameters

12x less than AlexNet



Inception Module

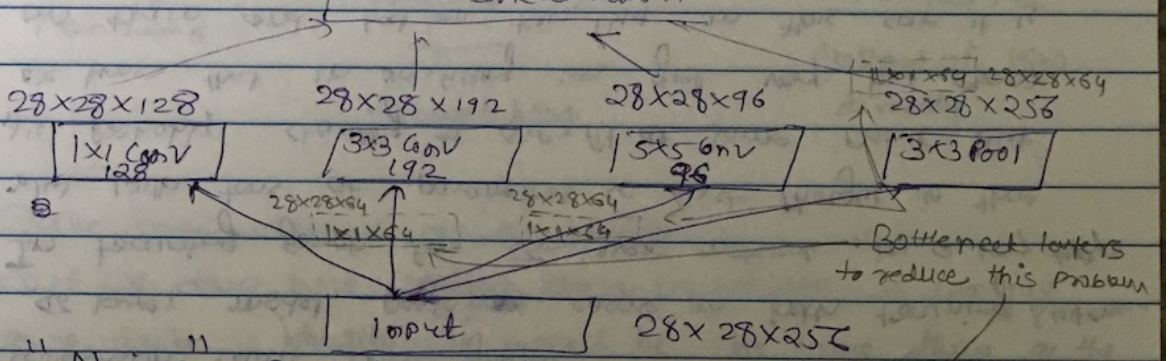
Good local network topology (network within a network) and stack these modules on top of each other

Apply parallel filter operation on the input from previous layer
- Multiple receptive field sizes for Conv (1×1, 3×3, 5×5)
- Pooling operation (3×3)

Concatenate all filters outputs together depth-wise.

Example $28 \times 28 \times (128 + 192 + 96 + 64) = 28 \times 28 \times 480$

~~$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$~~



Bottleneck layers to reduce this problem

Fig: "Naive" Module (Without dimension reduction)

Conv Ops [1×1 Conv, 128] → $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3×3 Conv, 192] → $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5×5 Conv, 96] → $28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total 854 M Ops !!!

Very Expensive.

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer.

Solution: "Bottleneck" layers that use 1×1 Convolution to reduce feature depth. (Image shown on previous page)

With dimension reduction in above image total no. of Ops - 358M

Bottleneck can also reduce depth after pooling layers.

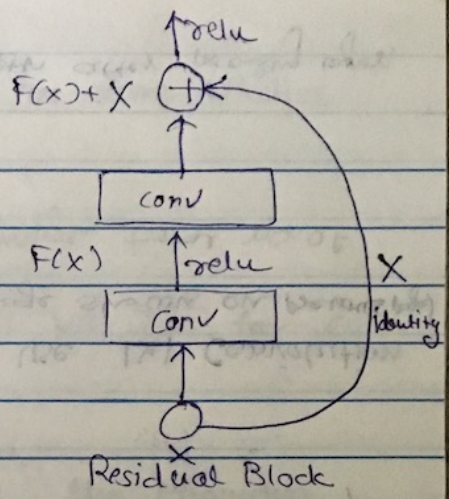
ResNet

Revolution of depth

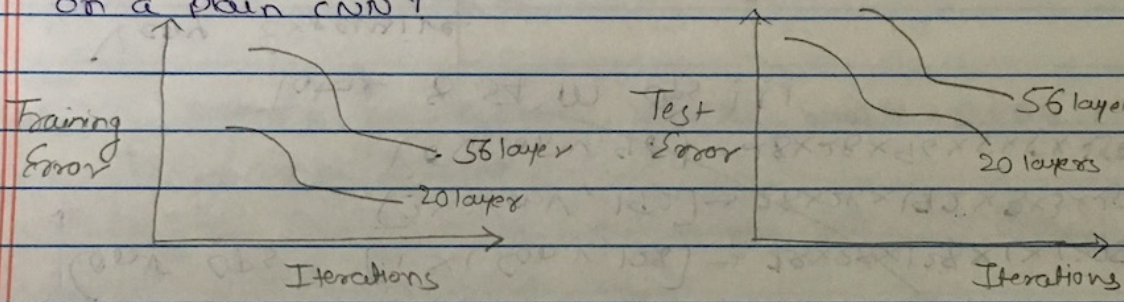
152 layers!

Very deep network using residual connections

3.57% top 5 error on ILSVRC'15



What happens when we continue stacking deeper layers on a plain CNN?



Conclusion from both graph

56 layer model performs worse on both training & testerror

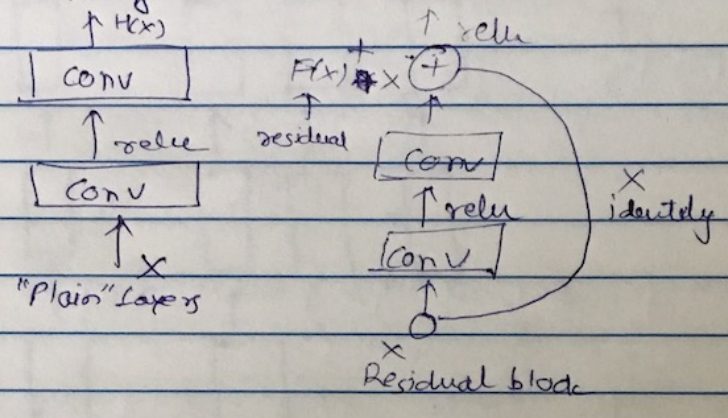
In training graph, for 56-layer network, its a very deep n/w with tons of parameters, so first thought in that it's probably starting to overfit at some point But we know that in overfitting we get very low error on train and bad on test. But in this case it is doing worse than 20 layers even on Training data too.

So, this is not caused by overfitting.

The problem is an optimization problem. Deeper models are harder to optimize

Solution: Use network layers to fit a residual mapping

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping.



Use layers to fit residual
$F(x) = H(x) - x$
instead of $H(x)$ directly

→ Take input, pass it as an identity
So if we don't have weight layers in between it was just going to be identity, it would be the same thing as the output, but now we use additional weight layers to learn some delta

→ Output will be → original x plus some residual (delta)

→ What we were trying to do is learn $H(x)$ but what we saw earlier is that it's hard to learn $H(x)$. So instead of $H(x)$ is equal to $F(x)+x$ and let's just try & learn $F(x)$

→ So, instead of directly learning $H(x)$, we are learning what needs to be add/subtract from input as we go to next layer

Studying this block for complete ResNet to make it 152 layers wide. For deeper net, they used "bottleneck" layers to improve efficiency

FYI

→ Batch Norm. after every CONV layers
→ Xavier/2 init
→ SGD + momentum (0.9)
→ LR: 0.1, /10 when validation plateaus
→ mini batch size 256
→ Weight decay of $1e^{-5}$
→ No dropout

Few other Network for study:

Network in Network (NiN):

Wide Residual Network → More width than depth

(ResNeXt.) → Parallel blocks in Residual path

FractalNet

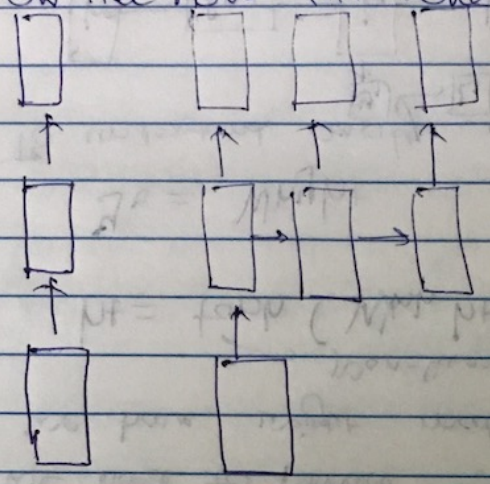
DenseNet → Each layer connected to every other layer in a feedforward fashion

SqueezeNet - Alexnet-level acc. with 50x fewer para. & <0.5mg Model size.

Lecture 10 RNN

June 17th 2018

Till now NN → One to one



One to One One to many many to one many to many

With RNN, we can do

Input is some object or fixed size image but now output is a sequence of variable length, such as caption where different caption has different number of words

This is an example of one to many

Many to One : Sentiment ~~Sigmer~~ Classification

Sequence of words (many) → Sentiment (one)

Video (many) (frames) → What activity in ~~video~~

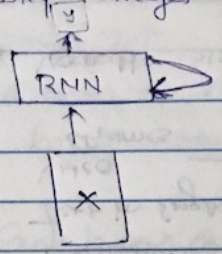
Many to many : Translation

Text in English → Text in French

Video (frames) → Classification of each frames.

RNN handles variable sized sequence data.

RNN can classify images by taking a series of "glimpses"



→ Core cell
→ Takes input x
→ Feed that into RNN
→ RNN has internal hidden state
→ Internal H.S. updated every time

that the RNN reads a new input.

→ And that I.H.S. will be then fed back to the model the next time it reads an input.

→ Usually want to predict a vector at some time steps (classify)

Recurrence formula

$$h_t = f_W(h_{t-1}, x_t)$$

new state ↑ some f^n w/ parameters W ; old state ; input vector at some time step.

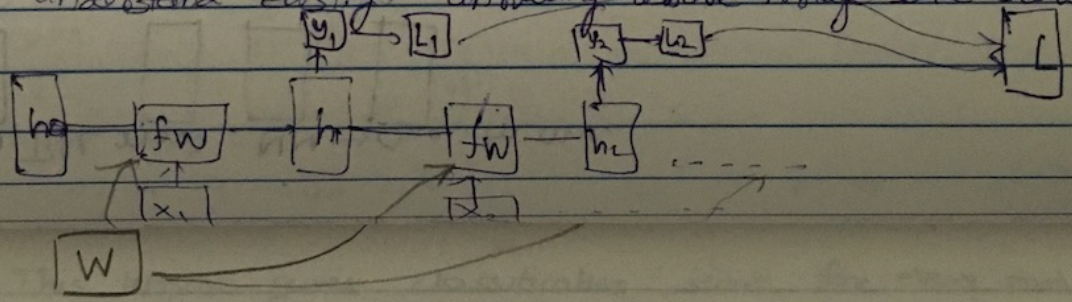
We need to produce the next hidden state
We have weight matrix W_{hh} and W_{xh}

→ Non-linearity

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

To understand easily, unrolling above image like below



REMEMBER: We are using same W matrix for all inputs

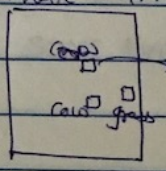
Lecture 11 06/27/18.

Detection and Segmentation

① Semantic Segmentation

Label each pixel in the image with a category label.
Don't differentiate instances, only care about pixels.

Idea 1: Sliding Window
Take an image or break it up into many small tiny local crops of the image and then imagine this as a classification problem.
Like, for this crop what is the central pixel of this crop?



Super Expensive in Computation.
to run fwd/bckwd pass
Terrible Idea!!!

Idea 2: Fully Convolutional Network
Design a nlw as a bunch of C. layers to make predictions for pixels all at once.

→ Rather than extracting patches of image, ~~we~~ can imagine having our nlw be a whole giant stack of Conv. layers with no fully connected layers so in this case we just have a bunch of Conv. layers that are all 3×3 with zero pad.
→ Pass an image to this stack. Final layer → C × H × W where C is the category we care about.
→ This tensor gives classification score for every pixel

What is upsampling & Logic?

1) In-network upsampling - Unpooling

Nearest Neighbor

1	2
3	4

2×2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

4×4

Bed of Nails

1	2
3	4

2×2

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

4×4

2) Max-Unpooling

→ Remember which element was max.

Max Pooling:

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

I/P 4×4

5	6
7	8

O/P 2×2

1	2
3	4

I/P 2×2

Max Unpooling

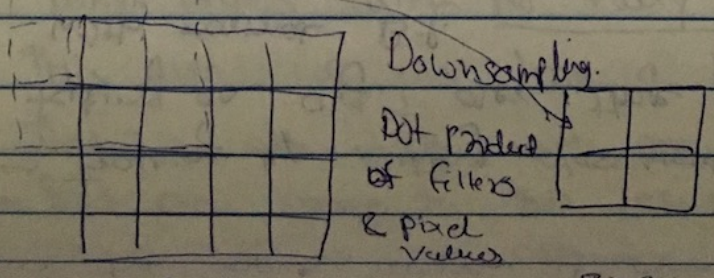
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

O/P 4×4

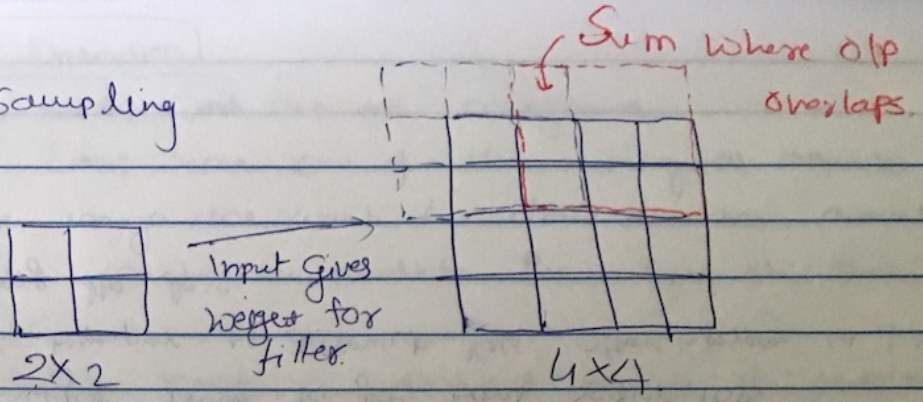
3) Transpose Convolution

All of above are fixed fn. Not learning.

T.C. → allow us to do learnable upsampling. w/ weig



4x4



Here we are going to multiply Scalar values (Cells in input) to the kernal (filter) value. Scalar values provides weights to the filter And then we copy those values to location.

Filter moves 2 pixels in o/p for every one pixel in the input

Stride gives ratio b/n movement in o/p & input.

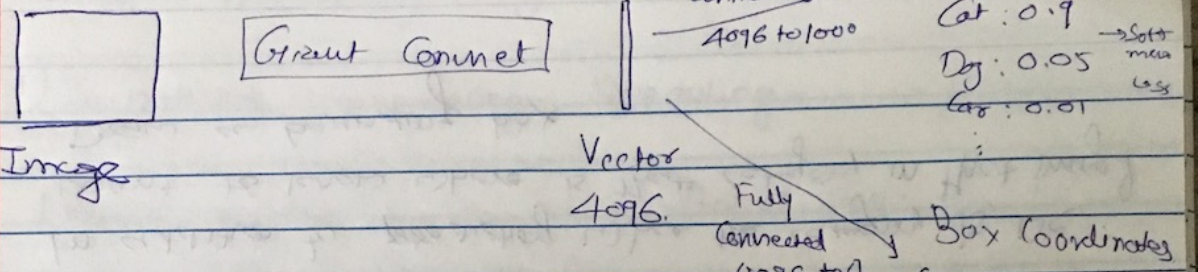
Other names: Deconvolution, Upconvolution, Fractionally Strided, Backward Strided Convolution

Final

~~Take image → Down Sample → Upsample → Cross entropy loss on each pixel → B.P. (Train)~~

Classification + Localization

In addition to ~~predicting~~ what the category is, we want to know where is that category in that image? Draw a bounding box



Treat localization as a regression problem!

NN will produce two different outputs.

One → class scores

2 → Four number for B.B.

@Training time, we will have two losses.

Assumption: Training images have both category & ground truth bounding box for that category

Softmax loss for category labels.

L2 Loss → Measures dissimilarities betn predicted B.B & G.T B.B

How loss (two) are affecting the gradients now?

→ We have two scalar loss. One more hyperparameter to weigeut them to get final scalar loss

→ This hyperpar is different than other hyperpar as it will change the loss.

Object Detection

Start with fixed set of categories.
Task: Every time one of those categories appears in the image, we want to draw a box around it and we want to predict the category of that box

Each image needs a different number of outputs!
For one object → 4 number for B.B
3 objects → 12 " " "

Approach 1: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as Object or background.

Problem: How do you choose "Crops"?
Have to choose many different sized crops.
Do not use!! Computation expensive.

App. 2 Region Proposals

→ Find "blobby" image regions that are likely to contain images
→ Relative fast to run. eg. Selective Search gives 2000 region proposals in a few seconds on CPU.

Rather than applying Classification n/w on every possible loc^n in image, instead first apply one of these region proposal n/w to get some regions where objects are likely located & now apply a conv. n/w for classification to each of those prop. regions

R-CNN.

→ ROI from a proposal method (~2K)

→ Regions have diff. sizes. To run them all on conv n/w for classification, size must be same.

→ Warp them to fix size.

→ Run each of them to ConvNet.

→ Classification decision for each of those.

→ Predicts BB. also

→ Inference (detection) is slow.

→ 47s/image with VGG16.

→

Fast R-CNN

Take image, Pass it to through ConvNet. Get feature map. Now, take crops (Rather than taking from image directly like in RCNN)